Ab Initio and Performance Tuning

24/10/2011

Hi-Tech ISU

Aarti Srinivasan
aarti.srinivasan@tcs.com

Ab Initio Overview

Ab Initio is suite of applications containing the various components, but generally when people name Ab Initio, they mean “Ab Initio Co>Operation system”, which is primarily a GUI based ETL Application. It gives user the ability to drag different components and attach them, quite akin to drawing.

The strength of Ab Initio-ETL is massively parallel processing which gives it capability of handling large volume of data.

Let’s componentise Ab Initio

1. Co>operation system
2. EME(Enterprise Meta>Environment)
3. Additional Tools
    3.1 Data profiler
    3.2 Plan-IT etc

Co>Operating System is ETL application; it comes packaged with EME. This is GUI based application. Quite simple is design due to drag and drop features, most of the features are quite basic and so learning curve is quite steep. Now it has further two flavor or sub classes:

1. Batch mode
2. Continuous flow

The both primarily do the similar things, but classically they are different in mode of processing as the name suggested. The “Batch Mode” is primarily used by most of customer gives the benefit of moving bulk data (daily/multiple times a day).

Continuous mode is more like “Click/Trigger” driven; say when you click on a web page the data flow starts, some of very large web based application run on Ab Initio server using Continuous flow.

EME is more like source control for Ab Initio, but it has many additional features like

1. Meta data management
2. Impact Analysis
3. Documentation tools

4. Run History Tracking
5. And surely, Check-in and Check-out

Data Profiler is data profiling tool, got the features for data quality analysis.

Plan-IT is primarily a scheduler built by Ab initio to run Ab initio jobs. It can be integrated with Ab Initio jobs.

For massive data processing, where time is of essence and performance and throughput is critical, AB INITIO stands head and shoulder above others.

1. Massively parallel Architecture
   - 1.1 Available data can be split and processed in parallel giving it huge processing advantage.
   - 1.2 Theoretically, it is possible to design a system using AB INITIO architect where any additional processing power can be achieved by adding additional resources in parallel, thus allowing any scale-up easy and possible.
2. Innovative component
   - 2.1 AB INITIO components like compressed indexed files and similar gives AB INITIO an edge when dealing with huge datasets. Though this component is not unheard of, in past, but AB INITIO implemented it successfully.
   - 2.2 Some new scripting features known as PDL(Program Definition Language) in AB INITIO allows flexibility, which is quite well received by AB INITIO developers and not easily available in other ETL tools.

Ab Initio software solves the toughest problems today facing developers and architects of enterprise-wide data processing systems: scalability, development time, metadata management, and integration.

Scalability
The Ab Initio Co>Operating System provides truly scalable parallel and distributed execution, allowing applications to tackle enormous—and rapidly growing—volumes of data. If you double the size of the data, it runs in the same amount of time; you just double the number of processors. If you need it run in half the time, again, you just double the number of processors. Applications built with Ab Initio software regularly employ more than a hundred processors to make quick work of processing the world's largest datasets. With the Co>Operating System, virtually any amount of computational power can be brought to bear on a problem.

Development Time
The Ab Initio Graphical Development Environment (GDE) enables truly rapid application development and enhancement. Through the GDE, one may build and modify the most complex applications in a

purely graphical way. This leads to very high productivity: in a handful of months, a small team can design, implement, and put into production a large data warehouse system using Ab Initio Software. Smaller systems can be put together in minutes.

Metadata Management

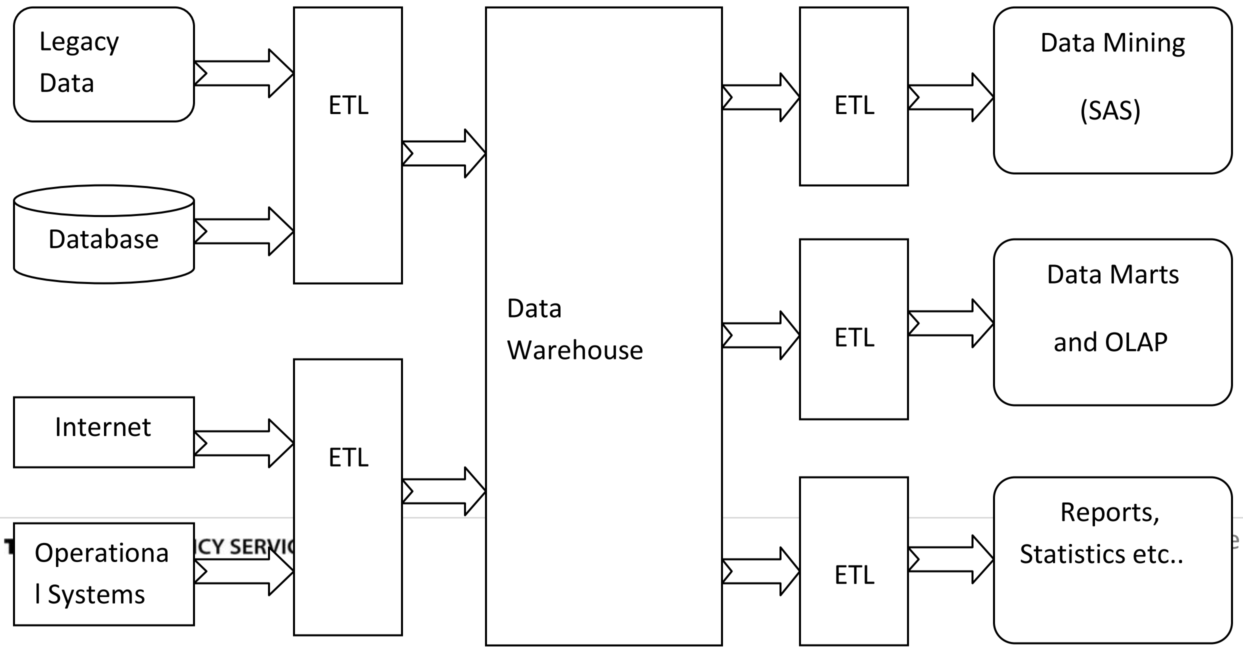
The Ab Initio Repository provides storage for all aspects of a data processing system, from design information to operations data. The Repository provides a rich store for the applications themselves, including data formats and business rules, enabling dependence analysis ("Where did this data come from?") and impact analysis ("If we made this change, what would be affected?").

Integration

Ab Initio software includes general capabilities for application integration. Using these capabilities, one can easily incorporate custom and third-party software packages into the Ab Initio component framework. There, they immediately benefit from the scalability provided by the Co>Operating System, the ease-of-use provided by the Graphical Development Environment, and the metadata management facilities of the Repository.

Applications of AB INITIO Software

Ab Initio software can pull data from practically any kind of source (legacy systems, operational systems, database tables, flat files, SAS datasets, the Internet, and so on), apply arbitrarily complex transformations to it, and move that data to practically any kind of destination. This flexibility makes it easy to integrate existing programs written in virtually any language into complete, end-to-end solutions. As you will learn in this tutorial, all of this can be done without programming. In a traditional data warehousing environment, Ab Initio has a place at both the "front end" of the warehouse, transforming and cleansing source data, and at the "back end" of the warehouse, extracting, transforming, and loading into other systems for analysis as shown in Figure.



Ab Initio software performs many compute-intensive operations like multi way joins and complex aggregations substantially faster than conventional software systems. Furthermore, Ab Initio's built-in scalability allows it to do all of these things in parallel across as many CPU's and servers as desired.
The flexibility and performance of Ab Initio software make it the ultimate in integration and scalability.

AB INITIO Solutions

AB INITIO software is used in a wide variety of systems, including:

1. Data Warehousing
2. Extraction, Transformation and Loading (ETL)
3. Real-Time Data Collection and Analysis
4. Customer Relationship Management (CRM)
5. Data Mining
6. On-Line Analytical Processing (OLAP)
7. Click-Stream analysis

and many more.

These systems employ:

1. Scalable, Parallel Execution of Existing Programs (written in C, Cobol, C++, Java etc)
2. Parallel Sort/Merge Processing, Aggregation and Referential Integrity Checking
3. Integration of Disparate Data, including Legacy data
4. Integration of disparate applications

Ab Initio Performance

Sometimes an ETL process runs considerably slow speed. During test for the small result set it might fly but when a million rows are applied the performance takes a nosedive. There can be many reasons for slow ETL process. The process can be slow because read, transformation, load.

For ETL process to be slow on read side here are some reasons :
- No indexes on joins and/or 'where clause',
- Query badly written,
- Source not analyzed.

What ETL performance is good enough?

Every ETL developer / Data Warehouse Manager wants to know if the ETL processing speed (Usually measured in NUMBER OF ROWS PROCESSED PER SECOND on target side)is good enough? What is the industry standard?

Usually for INSERTs, 1000 and for UPDATEs 300 rows/sec should be good enough. If you don't have special requirements then any average around these numbers should make you comfortable.

Select a Load Technique

An important step in an ETL process usually involves loading data into a permanent physical table that is structured to match your data model. As the designer or builder of an ETL process flow, you must identify the type of load that your process requires in order to: append all source data to any previously loaded data, replace all previously loaded data with the source data, or use the source data to update and add to the previously loaded data based on specific key column(s). When you know the type of load that is required, you can select the techniques and options that are available that will maximize the step's performance

After you have selected the Load style, you can choose from a number of load techniques and options.
Based on the Load style that you select and the type of table that is being loaded, the choice of techniques and options will vary. The Table Loader transformation generates code to perform a combination of the following tasks:
• Remove all rows
• Add new rows
• Match rows
• Update rows

Parameterization

One of the main advantages of using EME is 'Parameterization'. The ABI environment uses parameters to organize project and environment structure. Parameters are used to control the behavior of graphs and projects in a controlled and uniform way. BitWise development team strictly sticks to the standardized naming format for each of the parameters. The parameters are very efficiently divided as Team Level parameters, Project level parameters and Graph level parameters. The parameters that are common to all the process under one project are defined under Project level parameters. Thus it avoids re-defining of same parameter in the process. Adopting such kind of hierarchy makes the applications more maintainable. The standards followed also helps making graphs portable across multiple platforms and environment. The Application can be deployed with minimum coding changes to different operating system or any environment (Development / Test / Production).

In the 2.14 and higher versions of the product, BitWise has embraced and immersed into the Concept and and practical uses of psets (parameter sets). Elimination of code redundancy and re-use of the code is the prime pro of a PSET. PSET's are used within the EM>E for dependency analysis and data lineage, within job deployment to give a job name, and within the Standard>Environment to support the environmental features of graphs.

Scripting

Ab Initio and Unix scripting are tightly coupled. Scripting complements the Ab Initio environment by performing any pre-requisite tasks for the graph.

Ab Initio Capabilities

Since the ETL applications typically handle large volumes of data, special attention is paid to performance tuning. Some of the techniques used include:

- Performance Tuning and Reliability
  - » Setting up the environment variables with appropriate values Effective use of environment variables for File Assignments, Date/Time Assignments, Script Based Assignment and Constant Value Assignment.
  - » Providing information about the execution region (Development/Staging/Production) so variables can be populated and used accordingly.
  - » Performing clean-up and catalog activities using wrapper scripts.
  - » Dynamic Script Generation is the latest in Ab Initio world and one of its finest features, it has enabled the use of Ab Initio PDL (Parameter Definition Language) and Component Folding.
  - » Row Partitioning for parallel processing of the data
  - » Sort-Dedup, Filtering and Column partitioning to process only required data
  - » Checkpoints to provide re-run ability and reliability
  - » Phases to ensure optimum memory and CPU available for job execution.

Reusability

Reusability is an essence of all application design and architecture. Ab Initio provides reusability in form of sub-graphs and custom components to carry out common tasks across the job stream. XFRs in Ab Initio provide a kind of extension to reusability by containing common transformation logic at one place to be used by other graphs wherever required.

Code reusability was introduced and accomplished through Generic Load graph by Bitwise offshore Team. This graph loads source file to Teradata.The advantages of the generic graphs developed are:

- Reduction in development time for load process.
- Reduction in the number of objects maintained in the repository

EME for Source Code Mgt

The most influential feature of EME (Enterprise Metadata Environment) i.e. "Dependency Analysis" is followed for In-Depth analysis of time critical applications. The storage of graph related Metadata allows for data impact analysis to occur, giving the user a visual sense of how the data is changing in a graph, and the impacts those changes have on another graph.

Rudimentary EME features such as Version Control, Statistical Analysis, Code Sharing, Code Re-Usability, Data Lineage, Metadata Management and Code Promotion are productively trailed. For all applications, the standard directory structure is followed. Thus, the locations for storing the layouts, transformations, SQLs, etc. are fixed. This makes the applications more maintainable and it becomes easier for new developers to understand the application.

BitWise has developed strict coding conventions and guidelines based on its experience working Ab Initio and these are followed throughout all the projects undertaken. Three important Metadata goals namely DEFINE, BUILD & ADMINISTER and NAVIGATE are primarily achieved throughout all the projects undertaken.

Improving the performance of an existing graph

Working on performance problems in an existing graph is a lot like debugging any other problem. An important principle to follow when making changes to a graph, and then measuring what differences (if any) have occurred in the graph's efficiency, is to change only one thing at a time. Otherwise, you can never be sure which of the changes you made in the graph have changed its performance.

Controlling disk consumption

You can control the amount of space a dataset uses on a given disk by:

- Making it a multi-file and splitting its partitions among several disks
- Breaking the data up into smaller separate datasets
- Compressing the file

The COMPRESS and DEFLATE components can be used to compress data in a graph, and the UNCOMPRESS and INFLATE components to uncompress data. Compression saves space, although the time it takes to compress or uncompress adds to the graph's execution time.

Controlling memory consumption

You can reduce the amount of memory a graph uses by:

1. Reducing the number of components in the graph's largest phase.

   The peak memory consumption of a graph is determined by the memory consumption of its "biggest" phase, as explained in "Calculating a graph phase's memory usage". Making the phase smaller (by reducing the number of components in it) will reduce the amount of memory it requires; see "Calculating memory usage: An example", and especially "The effect on memory consumption of adding phase breaks".
2. Reducing the level of data parallelism.

   Each additional partition of a component requires space for the additional component processes in memory; reducing the number of partitions reduces the number of component processes.
3. Reducing the specified value of the max-core parameter in any component that has this parameter.

   The max-core parameter specifies the amount of memory you want the component to have accessible for doing in-memory operations such as sorts. Reducing this value is an obvious way to reduce a component's memory consumption
4. Taking advantage of sorted data by using components with sorted-input set to Input must be sorted or grouped.

   Components that process sorted input use very little memory compred to what would be required to sort the data.